Hackathon JLR: Task Assignment with Cost Minimization

Aditya Sridhar, Kshitiz Anand, Akshat Srivastava, Rahul Batish, Pranav Pable

June 2024

1 Introduction

This document proposes an approach for assigning tasks to workers that considers both skill compatibility and cost minimization. It leverages a cost matrix that captures the penalty associated with assigning a specific task to a particular worker.

2 Implementation Workflow

The following workflow diagram illustrates the overall implementation process:

1. **JIRA Data Extraction:** It begins by extracting all team issues and relevant details using a custom JIRA Automation API.
2. **Data Cleaning**: The extracted data undergoes cleaning to remove unnecessary words or symbols, ensuring accuracy for further processing.
3. **Skill Labeling**: Each issue is then labeled with the specific skills required to complete it.
4. **NLP Model Training**:
    - Fine-tuning: A pre-trained Sentence Transformer model is fine-tuned to enhance its ability to understand the specific context of our task descriptions.
    - Vector Encoding: Next, the cleaned text descriptions are converted into new vector encodings using the fine-tuned model.
    - Skill Classification: A Random Forest Multi-Class Classifier is trained on the labeled data to learn the relationship between text descriptions and skill requirements.
    - Skill Vector Output: The trained model generates the skill requirement vector for each task based on its textual description.

5. **Task Assignment:** Using the cost matrix and skill vectors, an algorithm assigns tasks to the most suitable employees, minimizing skill gaps and cost.

6. **JIRA Ticket Creation:** The assigned tasks are automatically converted into tickets using the JIRA Automation API, kickstarting the sprint

3 Skill Representation

Workers and tasks are represented using skill vectors. Each vector is a N-dimensional tuple, where N is the total number of skills considered. The value at each index signifies the proficiency level (e.g., 0.8 for high, 0.3 for moderate).

4 Cost Evaluation Function

The algorithm calculates the cost of assigning a task to a worker based on their skill levels. A vector is created according to each skill and each user is assigned a specific value for the skill. A similar vector is constructed for each story where each skill value represents the skill level required for the task. A typical skill vector would look like:

$$S_{vector} = [S_1, S_2, , , , S_k] \tag{1}$$

where Si represents the skill level for the ith skill according to skill taxonomy.

The cost is calcualted for the collection of stories and employees according to the formula:

$$C_{ij} = \sqrt{\sum_{k \in skills} max(0, Tj_k - Wi_k)^2} \tag{2}$$

where:

- C_{ij} represent the cost function if the ith person does the jth task.
- Tj_k : Skill level required for the jth story for the kth Skill.
- Wi_k: Skill level of the ith worker for kth Skill.

The function iterates over each skill index (k) and calculates the positive difference between the worker's skill level (Wi_k) and the task's requirement (Tj_k) only in case the required skill is less than the worker's skill.

The $\max(0, Wi_k - Tj_k)$ term ensures that if the skill level of the worker is more than that required, the cost is zero.

5 Cost Matrix Calculation

The cost matrix captures the cost of assigning each worker to every task. It is then used to calculate the cost of different permutations of assignment where we minimize the total cost for allocation.

$$C = [C_{ij}]_{i \in worker, j \in stories} \tag{3}$$

The function creates a cost matrix **C** with dimensions MxK. It iterates through each worker (W_i) and task (T_j), calculates the cost using 'find_cost', and stores it in the corresponding cell (C_{ij}) of the matrix.

This cost matrix becomes the foundation for assigning tasks to workers while minimizing the overall cost.

6 Skills Matching to Employee

Now that we have the cost matrix calculated, we have to assign tasks to each employee keeping in mind their bandwidth while reducing the cost function.

The following can be captured through the following constraint equations:

$$\sum_{j \in stories} \alpha_{ij} * sp_j <= BW_i \tag{4}$$

where:

- α_{ij} signifies if the jth story is assigned to the ith worker (1 if yes, 0 if no)
- sp_j represents the story points of the jth task
- BW_i represents the bandwidth of the ith guy

We are supposed to minimize to total cost, given by:

$$C_{total} = \sum_{i \in worker, j \in stories} \alpha_{ij} * C_{ij} \tag{5}$$

subject to the constraints:

$$\sum_{i \in worker} \alpha_{ij} = 1 \tag{6}$$

i.e. Each story is assigned to only one person

This optimization problem follows 0,1 multi knapsack problem which is a NP Hard problem and does not have a solution in polynomial time. We design an optimized algorithm where we use heuristic approach to greedily pick out the most feasible combination for the allotment of stories

Pseudo code for the algorithm implemented for task allotment:

```
Sort Tasks in decreasing order of story points
For task in story_list:
    best_employee_for_task = -1
    For employee in employee_list:
        if employee has bandwidth and is the best employee for the task:
            best_employee_for_task = employee
    if best_employee_for_task != -1:
        add current task to best_employee's task list
        bandwidth[employee] -=  task.story_point
```

7 Tuning the parameters

Over time we optimize and update the skill vector of all the workers using the metrics from the sprint such as the story status (if it is completed or in progress at the end of the sprint), the total story points completed which can be further used for a better allocation in the future.